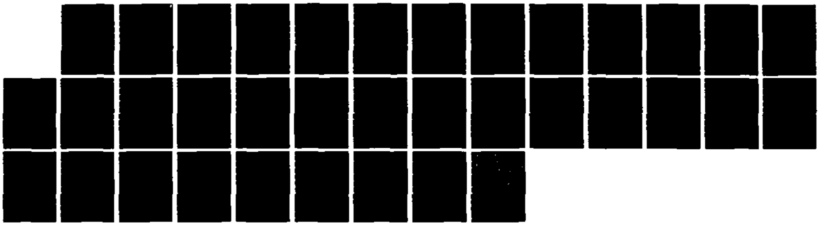
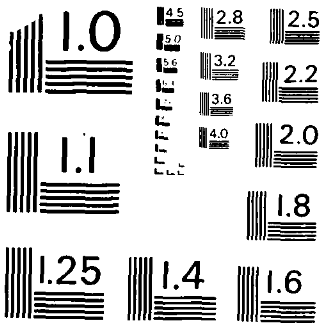
AD-A175 122 NATURAL LANGUAGE GENERATION: COMPLEXITIES AND TECHNIQUES(U) MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER AND INFORMATION SCIENCE D D MCDONALD APR 86 1/1

UNCLASSIFIED N00014-85-K-0017 F/G 5/7 NL




1.0
4.5
5.0
5.6
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

2

The Counselor Project

Department of Computer and Information Science

University of Massachusetts

Amherst, Mass. 01003

Natural Language Generation: complexities and techniques

David D. McDonald

April, 1986

CPTM #14

This series of internal memos describes research in artificial intelligence conducted under DARPA Contract N00014-85-K-0017

2

Natural Language Generation: complexities and techniques

David D. McDonald
April, 1986
CPTM #14

Natural Language Generation: complexities and techniques[1]

David D. McDonald
University of Massachusetts at Amherst
April 1986

This paper will appear in Nirenburg (ed.) **Theoretical and Methodological Issues in Machine Translation**, Cambridge University Press, 1986.

[1] Preparation of this article was supported in part by contract number SU353-9023-3 from Rome Air Defense Center, and contract number N00014-85-K-0017 from the Defense Advanced Research Development Agency, monitored by the Office of Naval Research.

Ultimately the success of any machine translation system rests in the quality of the text it produces as its output. Professional human translators, as a rule, must be native speakers of the language that they are translating into; otherwise they will not fully appreciate the nuances or idiomatic meanings of the word choices in the target language, or the inferences that are invited by alternative syntactic phrasings. If such choices are not made deliberately, with an appreciation of their consequences, then they are very likely to be made badly, and the sense of the original text distorted.

It would seem however that this fact is not well appreciated by the bulk of the machine translation community, since the generation programs that they employ are based largely on template-driven, fill-in-the-blank designs that have not been state of the art for more than fifteen years. Such programs have intrinsic limitations. They must deal with the language in fixed chunks, each with only minimal possibilities for variation and thus only the shallowest capacity to accomodate to their context. Large texts built from such rigid pieces have an awkward, mechanical style.

It is possible that such limitations do not pose a problem for MT, since the kinds of texts that it is presently important to translate: commercial notices, training manuals, etc., do not demand a high prose style or sophisticated knowledge of nuance. Yet if that is the case now, it will not remain so for long. Improving capacities in language analysis and representation will soon make it possible to deal with sophisticated texts such as newpapers or contracts, and the need for equal sophistication in generation will follow. The generators in MT systems will have to know how to convey whether a piece of information is new or old, whether it can be communicated tacitly by inferences from the choice of wording or phrasing or must be spelled out explicitly. There must be effective models of the choices a given language offers and their consequences if used. There must be ties to inference systems and to models of the common sense expectations of a reader of the target language. Above all there must be a rich understanding of the grammatical capacity of the language, of the dependencies between the alternatives it offers and how they may be navigated by the procedures of the generator as it constructs the output text. Simple generation systems do not have such abilities.

Artificial intelligence research in generation independent of MT has taken considerable strides however. It is in a position now to supply the generation capacities that strong MT systems will need. There is now a relatively sophisticated appreciation of what alternative designs for a generator are possible and of their consequences. Generation systems have been developed that are capable of quite delicate decisions of phrasing and content flow. Large syntactic grammars have been developed and are being exercised. Theories of the organization and construction of large texts and dialogs are being tried out. Text planning, rather than syntactic realization, has become the research frontier. Today's questions are: What model should a conceptual-level program have of the linguistic resources a language puts at its disposal? In what kind of theoretical vocabulary should it do its "linguistic reasoning"?

This paper examines the nature of generation systems today, the problems they have been designed to deal with, their strengths and their weakness. Its goal is to give the MT community a sense of what has been accomplished, and indirectly to show where MT researchers could consider adopting or adapting some of the AI work. This work on generation need not be done by AI people alone: MT can, for example, contribute to AI research on the planning-level by sharpening our collective understanding of the "carrying capacity" of the different parts of a language through cross-language comparisons that try to fit the ideas carried by the linguistic devices of a source language into the alternative devices of a target language. At lower levels, MT as a task can provide more linguistically demanding sources for generation than most any of today's expert systems. At the same time it is clear that generation is done for very different reasons in the two camps. The AI context is more like that of people dealing with each other in normal life--of which translation is not a customary part. Nevertheless, translation is a normal human capacity, and a considered comparison of the generation process in both contexts should tell us more about the nature of generation as a module within the human mind than could either by itself.

1. Introduction

The circumstances in which generation takes place during MT are different from those in a natural language interface. For the AI program using its NL interface, the purpose of generation is to deliberately construct a fluent and grammatical text that meets the communicative goals that the underlying program has specified. The MT program has no access to the goals the author of its source text may have had, and must either infer them or do without. In the interface, the generator works from a clean slate: it can choose whatever words, whatever syntactic constructions, that it believes will accomplish the goals. It also has a relatively free hand with stylistic matters such as sentence length or the complexity of the clause structure. The generator for the MT program, on the other hand, has the option of carrying over the wording and phrasing choices from the source text, and can follow the sentence length and complexity of the source as closely as it wishes.

These differences are both strengths and weaknesses for generation in MT. A weakness pragmatically because it may be difficult to adapt the processes and representations of highly skilled, interface-oriented generation systems to MT because of the very different presumptions about the kinds of structure-fixing information that are available. A weakness theoretically because MT encourages "early", cross-level transfer, i.e. taking words, or at a higher level syntactic constructs, from the source text over to their apparent counterparts in the target language at the moment they are recognized. This is the usual practice in MT (Slocum, 1984), since the alternative--waiting until entire segments of the text have been understood at a conceptual level, and the intentions of speaker recovered, then generating from those intentions--presumes that the MT system has access to a rich conceptual model of the subject matter of the text and a speaker's goals in talking about it, which is not at all the case in main-stream MT research.[2] Without knowledge of the intentions behind the use of a given linguistic device, the program is choosing the target language device blindly without the benefit of knowing its context of use.

The very same property can also be a strength, since it does, after all, make MT conceivable without the guidance of a conceptual model. Multi-topic, broad spectrum models with sufficient common sense to be of any use in a realistic MT domain are decades away from being realized (Lenat 1985). The cross-level, early transfer of constructions between languages may even be a considerable help if it is used with an eye towards identifying or constraining what the speaker of the original text's intentions may have been and thus how they can best be conveyed in the phrasing of the output text. (Remember that while the interface-based generator knows for sure what what the "speaker" that it is working for intends, an MT has no such privileged knowledge and must deduce it from what it reads.)

As a way of making it clearer what is at issue, it is worth looking closely at the nature of what today's generation program must accomplish in its normal behavioral nitch: what is the information flow; what implications does this have for its organization; what are the distinguishable components and the relationships between them.

1.1 Basic organization: generation versus comprehension

As a field, computational linguistics has far more experience in, and brings more sophistication to, the problems of language comprehension than of generation. There is a

[2] A good example of an MT system that does analyze the source text as thoroughly as the language comprehension system of an NLP interface is the recent work by Steve Lytinen at Yale (1984).

tendency therefore to look at generation in the same kind of terms as comprehension; a tendency that should be dispelled, since to understand the issues in language generation research one must learn to see it as a problem of construction and planning rather than analysis.

As a process, generation has its own basis of organization. Language comprehension typically follows the traditional stages of a linguistic analysis: morphology, syntax, semantics, pragmatics/discourse, moving gradually from the text to the intentions behind it. Generation has the opposite information flow: from intentions to text, content to form. What information is already known and what must be discovered is quite different than in comprehension, and this has many implications. The known is the generator's awareness of the underlying program's intentions, its plans, and the structure of any text the generator has already produced. Coupled with a model of the audience, the situation, and the discourse, this information provides the basis for making choices among the alternative wordings and constructions that the language provides--the primary effort in constructing a text deliberately.

With its opposite flow of information, it would be reasonable to assume that the generation process can be organized like the comprehension process but with the stages in opposite order, and to a certain extent this is true: pragmatics (goal selection) typically preceeds consideration of discourse structure and coherence, which usually preceed semantic matters such as the matching of concepts to words. In turn, the syntactic context of a word must be fixed before the precise morphological and suprasegmental form it should take can be known. One must however avoid taking this as the driving force in a generator's design, since to emphasize this ordering of linguistic representational levels would be to miss generation's special character, namely that **generation is above all a planning process**. Generation entails realizing goals in the presence of constraints and dealing with the implications of limitations on resources, e.g. the specific expressive capacities of the devices a given language happens to have, or the limited space available in a sentence or paragraph for expressing ideas given the constraints of the prose style that has been choosen.

Once one accepts the view that generation is planning, it is natural to organize its particulars around the efforts involved in making decisions: decisions to use certain words or syntactic constructions, decisions to post constraints on later decisions. The criteria for the division of the process into components now follow from the character of the decision-making involved, which will vary along the following three dimensions: (1) What information does the decision draw on: Properties of lexical items? Conceptual attributes? Details of planned but not yet realized rhetorical structures? Details of the text's surface structure? (2) What is the decision dependent on? What other generation decisions, if made differently, would force a change in this decision? If the generation process is to be indelible (i.e. never retracting its decisions), then this dependency structure will have to be respected in the order in which individual decisions are made. (3) How should the decision's conclusion(s) be represented? Does a conclusion dictate linguistic actions or just constrain other decisions? Can it be acted on immediately, or must it be scheduled for execution at a later moment and if so, how is that moment described?

The decision-making perspective as an organizational criteria for generation yields an ordering on events based principally on what dependencies govern a given decision and what representations will supply any needed reference information. In most cases this tends to reinforce a breakdown into stages like the ones identified by linguists, but only because that is a natural order in which to make the decisions.

2. TERMINOLOGY

Before proceeding further, it will be important to establish a working vocabulary for describing the parts of the generation process. To begin with, there are the informal terms that are constantly used for effect if not for specific meanings: "higher", "later" and other vague terms of that sort. There is a fairly consistent temporal orientation that generation researchers have informally adopted, and a corresponding spatial orientation, though it is less established. Intentions and concepts, the starting point for generation in an interface, are "earlier" and "higher"; linguistic matters are "later" or "down" from the concepts, with the conventional linguistic derivational geography: discourse, semantics, syntax, morphology, phonetics and speech, running from higher to lower in that order when they are discussed as a linear progression. Higher entities are more abstract, lower ones more concrete.

The question of what goes into a generation "system" or a "generator", or of where to draw the boundaries between components will be answered differently depending upon who one talks to. More often than not it varies simply according to what aspects of the problem a research group has chosen to work on simultaniously. Some projects take "the generator" to encompass all of the processing that is not domain-specific reasoning (Mann & Moore, 1981), others are comfortable restricting the term to cover only the linguistic processing, omitting the establishment of goals and the planning of content and large-scale organization (McDonald, 1983). This appears to be an issue of taste and the structure of research projects rather than one with technical consequences. Bearing in mind then that not all researchers in generation would agree with this breakdown, the rest of this section briefly introduces the processing elements that take part in generation, proceeding from early to late in the process. In the rest of this paper, references to "the generator", when not otherwise specified, will be to the linguistic processor that is responsible for realization.

When the generation system is part of a man-machine interface it will have some relationship to the other major component in that interface, the language comprehension system. Some kind of bridge, a representation of the discourse as a whole, must be maintained to insure that the information is available to support cohesive linguistic choices and interpretations across the turns of a dialog. Managing this representation and directing the overall communication activity should be some kind of discourse controller (see for example Bruce, 1975; Wilensky et al., 1984; or Woolf, 1984). In most ordinary cases, however, this controller will not be a separate module but just the executive portion of the underlying program, the non-linguistic, domain-oriented program that human users employ the interface to talk to. What this underlying program is: a cooperative database, expert diagnostician, ICAI tutor, commentator, apprentice, advisor, presently is of no significant influence in how the generation system is designed.

MT systems of course will usually have no counterparts to any of these "supra-generation" modules. They do have the pragmatic equivalent of the "urge to speak" by which those modules set the process in motion; the question is whether what then ensues involves processing of the same kind, and then whether it is profitably broken down into the same kinds of stages.

Within an interface, the generation process starts when some event within the underlying program leads to the need for it to speak: perhaps to answer a question, to give an explanation or pose a question of its own, or in a sophisticated underlying program, perhaps to interrupt the user's activities in order to point out an impending problem. Once the process is initiated, three activities take place: (1) identifying the goals the utterance is to achieve, (2) planning how the goals may be achieved by evaluating the situation and available communicative resources, and (3) realizing the plans as a text. For a computer, goals would usually be to impart certain information to the audience or to prompt them to some action or reasoning. People, of course, talk for social and psychological reasons as well as practical ones, but this need is beyond the ken of today's computer programs. Planning involves the selection (or deliberate omission) of the information to be conveyed directly in the text (e.g. concepts, relations, individuals). A coordinating rhetorical framework or schema for the utterance as a whole must also be adopted (e.g. temporal progression, compare-and-contrast). Particular perspectives may be imposed to aid in signaling intended inferences.

The non-linguistic plan or specification is typically called a message. The mental image often evoked by the term "message" is of written notes passed from one person to another, for instance as the result of a telephone conversation; however this image is much too simple. Researchers who study both planning and realization continually make the point that there is no clean boundary between the two activities (see for example Appelt, 1980 or Danlos, 1984): Planning proceeds by progressive refinement and must appreciate the linguistic consequences of its decisions. The realizations of the largest, most encompassing relations in the message create a grammatical context that imposes constraints on what can be planned for their arguments. Goals may emerge or change in priority opportunistically as planning and even realization proceeds. Thus though it is usually profitable to think in terms of an abstract "message" that defines and controls what the later linguistic processing is to produce, one must be careful not to presume from that that the generation process divides into two distinct stages or even that the message is constructed all at one time before any actual text production has begun.

Realization is the process of manifesting the planner's directives as actual text. This process has a counterpart in every MT system, since the equivalent of these directives will be the system's model of the source text being translated. Realization depends upon a sophisticated knowledge of the (target) language's grammar and rules of discourse coherency; most realization components typically maintain a syntactic representation of the text as it is being assembled. Usually it is only the realization process that has any direct knowledge of a language's grammar . The form this knowledge takes is one of the greatest points of difference between generation projects, though all projects largely agree on the function it should serve.

The purpose of grammar in generation is very different than in traditional linguistics. There a grammar is a body of statements in a notation, with the chief interest of the theoretician being on the structure and expressive capacity of that notation rather than the particulars of what rules a given language needs. **In generation the function of a grammar is to define and constrain linguistic choices.** Choices are dependent on the goals and situation; consequently the information in the grammar must be tied to some kind of model of language use. Generation researchers ask what circumstances lead to the choice of one alternative over another--what functions do the various constructions of the language serve that will make them able to fullfill a given goal or fit a given discourse situation.

3. WHY DOES GENERATION SEEM TO BE A SIMPLE PROCESS?

A generation system can be no better than the underlying program it is working for. If the program has only simple thoughts, the generator will only be able to say simple things. Measured in terms of the inferential content and structural relations that would make a text interesting linguistically--i.e. a design challenge to the researcher on generation--nearly all candidate underlying programs "think simple thoughts" even today.

It is then no accident that until the early 1980's generation was considered by most people in AI to be a relatively simple problem. The level of skill that generators had to exhibit was minimal because only minimal demands were made on them. Taking a statement in an internal representation of the sort people used in the middle 1970's, say `(#supports :block6 :block3)`, coupling it with attributes stored separately for the individuals, and producing "*The big red block supports a green one*" requires such a simple computational apparatus (typically a technique known as "direct replacement", see below) that the design is usually not worth publishing today.[3]

Output from Anthony Davey's program PROTEUS (1974), commentating a game of Tic Tac Toe that it played with a user.

"The game started with my taking a corner, and you took an adjacent one. I threatened you by taking the middle of the edge opposite that and adjacent to the one which I had just taken but you blocked it and threatened me. I blocked your diagonal and forked you. If you had blocked mine, you would have forked me, but you took the middle of the edge opposite of the corner which I took first and the one which you had just taken and so I won by completing my diagonal."

The monologue produced by John Clippinger's program ERMA (1977) modeling an actual psychoanalytic patient talking to her therapist. Text segments in parenthesis are what ERMA was planning to say before it cut itself off and restarted. This is an actual paragraph from a transcript of the patient reproduced in every detail.

"You know for some reason I just thought about the bill and payment again. (You shouldn't give me a bill.) <Uh> I was thinking that I (shouldn't be given a bill) of asking you whether it wouldn't be all right for you not to give me a bill. That is, I usually by (the end of the month know the amount of the bill), well, I immediately thought of the objections to this, but my idea was that I would simply count up the number of hours and give you a check at the end of the month."

FIGURE ONE

When the people developing the generator have the opportunity also to develop the underlying program and thereby to insure that it can supply interestingly rich perspectives and intentions, a considerable level of sophistication can be achieved in the text. Figure One shows examples of the output from two such cases, both of them developed around 1974. Anthony Davey's Tic-tac-toe program was a player as well as a commentator. It had a rich conceptual model of the game, and consequently could use terms like "*block*" or "*threat*" with assurance. Its heuristics for grouping the descriptions of moves together into complex sentences were based on notions of salience and tactical consequences. John Clippinger's emulation of one paragraph of speech by a psychoanalytic patient was the result of a computationally complex model of the patient's thought processes: from the first identification of a goal, through planning, criticism, and replanning of how to express it, and finally linguistic realization. Clippinger's program had a multiprocessing capability--it could continue to think and plan while talking. This allowed him to develop a model of "restart" phenomena in generation including the motivation behind fillers like "*uh*" or dubitives like "*you know*".

But such "double work" in a research effort is unusual, and work on generation has suffered as a result of the high costs of getting a research effort started. In working on parsing it is clear what one begins with--the words of a text--and one can chose to recover as little or as much of the text's semantic and pragmatic analysis as one likes. Generation on the other hand requires one to take a stand on the nature of conceptual representation and

[3] This example comes from Terry Winograd's 1970 thesis (see Winograd, 1972), where the amount of code for generation was only about 2% of what went into comprehension, and virtually none of it supported a model of grammar. (Pronouns were introduced into output by reparsing the output and doing substitutions.)

intention before one can even begin. To start from anything less abstract, for example to take the input to be a hierarchical structure of verb-like relations over noun-like objects (such as one might get from a composition-driven semantic interpretation component), is to see generation as just a matter of linearization plus local grammatical realizati on rules.[4]

When one looks at the apparent capabilities and knowledge that people have about their own generation, it becomes difficult to believe that the process or its input could be that simple. Consider the versatility of form and use that the native human speaker can make use of in our simple example. There was no grammatical requirement to say "*a green one* ", and not "*a green block*": a contextual influence, presently not well understood, determines this alternation, taking into account the possible ambiguity of the pronoun on the one hand and the cohesion effect that it brings on the other. More obviously, we can ask why the sentence should be couched as a statement about the red block (its subject) rather than appearing in the passive mood as a statement about the green block. Mood is not a feature that can be decided in isolation from the context of what else is being said and the intentions behind it--it can require reasoning. For example since the option to use either active or passive is not available with every verb or set of adjuncts, this choice can influence the other, more abstract choices that feed it.

Native speakers have knowledge about the uses to which phrases can be put that is beyond the ken of simple generators that do not incorporate a grammar designed to support the kind of linguistic reasoning that generation calls for. Consider wanting to use the Support assertion as an attribute of the green block, for example as a way to distinguish it from the other green blocks: "*...the green block that's supported by the big red one*". How does the generator represent to itself in a general way the fact that the relative clause is available as one of the realizations of the assertion? What gives the planner that feeds the generator assurance that such a usage is possible in this case? Can the generator use its representation of the syntactic structure of the declarative form of the assertion in producing the relative clause, or should it have an independent linguistic origin?

These are not simple questions to answer. Furthermore it is not even possible to begin without adopting an anchoring framework for the generation process as a whole by which to coordinate the individual answers. Yet the "startup" costs on any such framework are high: the way information and decision making flow through the process makes it very different in intrinsic character from the other language processes one might look to for direction. When coupled with the further difficulty of finding a strong underlying program to work with, the net effect of these problems has been that until recently very little work on generation has in fact been done. Few people worked on generation during the last decade (or stayed with the problem for more than a year or two) either because they found the task too simple to be interesting (when working forwards from the sorts of texts that reasoning programs needed at that time), or because they found it too difficult to make any headway (when working backwards from the complexities of actual human texts).

[4] One of the earliest efforts in generation, that of Bob Simmons and Jonathan Slocum (1972), saw the problem in just those terms. The input to their system was a semantic network structured by case-frame relationships such as "theme", "locus" or "mood"; its output was single, simple sentences like "What did the merry widow dance?". As pointed out by Bonnie Webber (1971), the surface form of the output was latient in the already linguistic vocaoulary that they used in their input representation; all that remained was to read the structure out and make the necessary morphological adjustments.

For this to be a reasonable general approach, the underlying program must use these same case-based networks in the internal representation that it uses for reasoning, in effect "thinking in English". If that is not the case (and it rarely is), some kind of planning and selection facility must mediate between the internal representation and the input language to the generator. Neil Goldman (1974) did just this as he generated paraphrases of expressions in conceptual dependency representation: he used a set of discrimination nets to chose lexical heads and subcategorization frames, which he then expressed in Simmons and Slocum's notation for realization by their generator.

Today there are candidate underlying programs, available from other research efforts, that are sophisticated enough in their reasoning to need sophisticated generators to voice their thoughts. As a result, the startup costs of good research on generation are more easily paid and more people are entering and staying with the area. This same increase in program sophistication is making other researchers in AI more appreciative of the need for careful generation because they see the difference that a slight change in wording can make and realize that perspective and context need to be considered in what their program says--they are looking for new approaches to generation to replace their template-based standbys. Before looking at the new work however, it is important to have a clear understanding of how the old technique works.

5. THE DIRECT REPLACEMENT TECHNIQUE

"Direct replacement" is the name that generation researchers have given to the informal, "natural" technique for text generation from internal representations that non-generation researchers tend to use (see Mann et at., 1982). It is a more informative term for what is informally known as "using templates", and has been independently rediscovered countless times. One of its major benefits, often overriding its deficiencies in grammar and text construction, is that it allows the programmer to use expressions directly from the internal representation of his program, thereby eliminating any need for a text planner or explicit representations of goals. Good descriptions of how the technique has been used in specific projects can be found in Swartout (1981), Chester (1976), or Forbus and Stevens (1981).

In examining any approach to realization--the part of generation most concerned with linguistic matters--there are three things to be considered: (1) What is the knowledge of natural language grammar: how is it represented; how is it brought into play during the processing? (2) What is the control structure: what determines what action these systems will take next; what possibilities for variation are there, and how are they embodied? (3) How does the approach organize the knowledge of how to realize specific objects (relations, attributes, etc.) from the underlying program: how is it embodied; how is it incorporated into the control structure?

As an example, we can look at the what was involved in the production of the sentence below, which is extracted from the standard dialog of Terry Winograd's SHRDLU program (1972). Because of its use of pronouns, SHRDLU was a bit more fluent than most direct replacement generators are programmed to be, but it is otherwise quite typical.

(Said in answer to the question "*How did you do it?*", which referred to an earlier directive to "*stack up both of the red blocks and either a green cube or a pyramid*".)

SHRDLU: *"By putting a large red block on the table; then letting go of it; then putting a large green cube on it; then letting go of that cube; then putting the red cube on that cube; then letting go of that cube."*

SHRDLU's generation knowledge, like the rest of its linguistic knowledge, was represented procedurally. Specialist programs or code fragments would be associated with terms in the internal representation by attributes indicating what they were to be used for. Special "data driven" interpreters would dynamically combine these fragments into full programs as the situation warranted. SHRDLU's generation program for the event type #puton, for example, is this fragment of Lisp code:

```
(append (vbfix 'put) obj1 'on obj2)
```

While inelegant by today's standards because it calls Lisp functions directly ("append") rather than employing an abstract schema, this specialist program is never the less illustrative of the genre. The underlying message for the example text is the list below of events extracted from *SHRDLU's model of its history*. This list is taken directly from the model; the individual assertions are in the very same representation that SHRDLU used for its internal reasoning.

```
(#puton :B6 :table)

(#ungrasp :B6)

(#puton :B3 :B6)

(#ungrasp :B3)

(#puton :B1 :B3)

(#ungrasp :B1)
```

To produce the text, each of the the assertions in the message, e.g. `(#puton :B6 :table)`, is interpreted as though it was an instruction in a special kind of program. The term `#puton` becomes, in effect, a function call whose arguments are the objects `:B6` and `:table`. The specialist generation program just shown is `#puton`'s definition, and the arguments will be bound to the program's formal parameters `obj1` and `obj2`. The Lisp function `vbfix` ("verb fix") is one of SHRDLU's internal routines. It is responsible for specializing the form of the verb to the grammatical context. The model of context that SHRDLU used was about as simple as one could imagine: when answering a "how" or a "when" question use the ending "+ing"; when answering "why" questions use the infinitive.

The generation process consists simply of the interpreter scaning the message from beginning to end as a stream, replacing expressions and subexpressions directly with the values returned by their generation functions. No intermediate representations mediate between the internal expressions and the words--hence the name "direct replacement". For example, when the generation process has finished evaluating the second element of the first expression in the list, it will look like this:

```
(append '(putting)
        '(a large red block)
        'on <== next expression to evaluate
        :table )
```

Literals like "*on*" are already words and are unchanged by the process; the sentence initial preposition "*by*" is introduced directly in the same way. Terms from the model like `:B6`, and in a moment `:table`, are replaced by lists of words assembled by the specialized generation program the programmer has associated with them. Embedded function calls like the one to `vbfix` are executed as they are reached, and are expected to return words. A generation interpreter like this is easy to write: with a good programming environment a competent Lisp or Prolog programmer can turn one out in an afternoon.

Returning now to the original questions: knowledge of grammar, control structure, and the realization of specific object types, we have seen that with direct replacement there is no explicit model of natural language grammar. Grammatical relations are manifested in the output text ad hoc by specialist routines that are incorporated into the replacement stream explicitly whenever they are needed. This piecemeal, directly programmed approach to grammar is effective enough in the construction of simple sentences, but breaks down as the complexity of the grammatical relations increases. The difficulties of embodying the grammatical situation in a programs' state without the mediation of an explicit linguistic representation eventually require a programming tour de force to overcome and lead to a design that is unwieldy and awkward to extend. The pragmatic limit in grammatical complexity is using a subordinated clause: the effort required to adapt a declarative sentence to be a relative clause, or to form "*Harry wants to go home*" from a composite internal expresion like `want((Harry, go-home(Harry))` is too much for generators without proper grammars

Control is vested directly in the conceptual representation. As just illustrated, messages are built from expressions in the underlying program's internal representation and are treated as executable generation programs. The natural structure of the expressions defines the order in which their elements will be evaluated and the phrases of the text will appear: a list will be enumerated element by element; a structured object (e.g. a frame or a database record) might be read out field by field.

This control technique is convenient and immediate. There are no wasted actions, and no need to define a control structure somewhere else in the generator's design. Its drawback is that the organization of the output text is in lock-step with that of the message. This enforced isomorphism is an inescapable consequence of the design. It is not troublesome if the underlying program's internal representation has been designed with natural language in mind (see for example Swartout, 1981). However it is problematic in two cases: (1) where the same internal expression can be viewed from several different perspectives depending on the situation, with a different organization of the text appropriate in each case; and (2) when it would be best stylistically to combine several independent expressions into one cohesive linguistic unit.

The knowledge of how to choose words to realize individual objects or relations is vested in the set of specialist programs, usually one for each expression type. SHRDLU's generation specialist for blocks is typical. It was organized as a function from an individual block (say `:B6`) to a list of words, and used a very simple algorithm that is the model for many other systems: start with the common noun that names the kind of the object ("*block*"); add enough of the object's attributes to distinguish it from all the other such objects in the scene. Attributes that can be realized as a single adjective like size or color are placed before the common noun; relations like support are placed afterwards; the determiner "*the*" is placed at the front of the word list to indicate that the phrase refers to an individual object. The list returned as the realization of `:B6` would be roughly as shown below.

```
( 'the  (#size :B6 (200 200 600))  (#color :B6 #red)  'block )
```

In fact SHRDLU's actual list was more ad hoc, since its generation procedure had no way to distinguish an occurance of an assertion like `(#color :B6 #red)` that was intended as a description of `:B6` (and consequently should be realized as just an adjective) from one intended as an independent statement (and realized as a clause). This is an endemic problem in direct replacement systems, and stems from the central aspect of their design, namely that realization goes directly from the conceptual representation to words without employing any intermediary structure. If one added to the block specialist's output list an annotation giving the perspective under which the two assertions were to be viewed, then that annotation could serve as the basis for distinguishing the adjective realization from the clause, and most of the problem would be taken care of. Annotating an expression inorder to describe to the generator its intended use provides a context for its realization. Without a distinguishing context any generator will be forced to always realize a given object in the same way each time (barring deliberately random variations).

The relation that prompts the mention an object in the first place, for example the `#puton` event with its reference to the objects `:B6` and `:table`, also introduces a context that can be relevant in deciding the object's realization, one that the generation process should not overlook. A subtle deficit of typical implementations of direct replacement is that they bar any possibility of representing this context because of the timing of when they realize a relation compared to when they realize its arguments: A practically automatic design choice in building any special purpose evaluation function is to follow the protocol used by Lisp and other main-stream functional programming languages, "applicative order". In this protocol, the arguments to a relation (function call) are evaluated before the relation is. In the present case this can mean that, e.g., the final forms of the noun phrases are choosen before the linguistic form of the verb can be known. This is one of the reasons why all but the simplest grammatical forms can be difficult for a direct replacement generator to produce, since the generator is committed to the wording for embedded elements before it can know what grammatical constraints the embedding relation is going to impose. We will return to this issue later in the section on description-directed generation, where an alternative, "normal order", evaluation protocol makes this sort of context available in a natural way.

5. MODERN APPROACHES TO GENERATION

Natural language generation is becoming an increasingly popular research topic: There have been a significant number of Ph.D. theses presenting artifical intelligence approaches to generation in recent years: McDonald 1980; Appelt 1981; McKeown 1982; Tait 1982; Conklin 1983; Hoenkamp 1984; Danlos 1984; Kukich 1985; Jacobs 1985, as well as some closely related theses on problems involving text planning: Clancy 1980; Swartout 1981; Woolf 1984; McCoy 1985.[5] In addition, three international workshops on generation have been held in since 1983, with the 1986 workshop in Nijmegen expected to have more than 60 participants, and there have been full sessions on generation at recent national and international artificial intelligence conferences (two sessions at IJCAI-85).

Generation is emerging as a recognized research area. Its practitioners are young, vigourously engaged with their work, and strongly mutually identifying. The intellectual climate within the generation community is not unlike that of the language comprehension community of about 1974, with a roughly similar number of players and a similar feeling in the air that significant things are happening. Unfortunately, as with comprehension in 1974, it is presently very hard to sort the different research efforts on generation into coherent schools of thought. This is in part because even now there are not so many researchers following any given approach that it can be properly called a "school" (as for example the use of ATNs versus lexically activated deamons typify different schools in language comprehension research). A confounding problem is that individual generation systems are unusually hard to compare, again because generation starts with an empirically unknown representation: the *thoughts*, *conceptualizations*, and *intentions* of the human mind. Different projects inevitably work from different representations and focus on different technical problems. This lack of a common starting point has made it hard for researchers to build on each other's work, or even to replicate each other's examples. Nevertheless the commonalities are there, and can be expected to be more salient in the future.

One common point is an overriding concern with two matters: the diversity of forms in natural languages, and how the generation process should be controlled. Diversity of form is a matter that is easily overlooked in studies of language comprehension. There, two ostensibly synonymous alternative forms will end up converted to the same "cannonical representation" because the differences between them will play no role in the reasoning of the underlying program. To the generation researcher, however, it is a nagging problem. Whenever the grammar defines an alternative, as for example between active and passive versions of the same clause, between the quantifiers "*some*" and "*any*", between expressing references to natural kinds as bare plurals ("*whales*") versus false definates ("*the whale*") versus prototypes ("*a whale*"), and on and on, the realization procedure must make a choice. Omitting the alternatives is both dull and inappropriate, since no surface alternatives are truely synonymous in all contexts (i.e. the choice without consequences for what is communicated), and crutial distinctions can be lost if some alternatives are arbitrarily not made available. The generation researcher's problem is to learn when and in

[5] The previous spate of theses on generation was a full six years earlier in 1974, with Clippinger, Davey, and Goldman. Their research built directly on the first rush of significant results in natural language understanding a few years before, specifically the work of Winograd (Clippinger and Davey) and Schank (Goldman).

what way(s) differences are consequential and how they are governed by situation and the speaker's intent.

Approaches to the question of choice are inexorably tied up with approaches to control. Which of two alternatives the generator picks has obvious consequences for what actions it takes next, and the way in which linguistic knowledge is represented has consequences for how alternatives are defined and when they are considered. Every surface text, however small, is the result of dozens of choices, and it is not reasonable, at least not from an engineering standpoint, to combine these choices into massive single discriminations. This leads to the question of how to order choices and how to represent the intermediate results. What awareness does the generator have of the dependencies between choices? How are these dependencies represented and made to influence the control algorithms?

These will be the themes for the rest of this paper as we look at the different approaches that are now being taken to the representation of grammar and the organization of the realization process. The earlier stages of generation, the identification of goals and the planning of what should be said, will not be discussed here, partly because there is not nearly as much commonality in the approaches or even the themes of approaches in text planning as there is in realization, and more because it is difficult to discuss goals or planning technique separately from the underlying program being worked with; a proper treatment thus would take the discussion too far afield.

6. Mixing control with the statement of the grammar

The first question to be asked when looking closely at a particular approach to grammar in generation is the relationship of the grammar to the rest of the realization process. Logically it would seem that dealing with the information content of the message should take priority over accomodating the stipulated regularities of a language's grammar. This is the stance taken in the direct replacement approach, where realization routines call grammatical functions when they need them (recall `vbfix`), but of course that approach has severe limitations in its capacity for linguistic expression, making it suspect. (The grammatical complexity of even standard newspaper prose is well beyond what can be programmed without an explicit grammar.).

By the same token, many of the details that make a text grammatical arguably do not and should not be expected to have any counterparts in the thoughts of any underlying program and thus must be independently supplied. Person and number agreement of subject and verb are an obvious case in English, as are relative pronouns (e.g. "*who*" vs. "*whom*"), the infinitive marker "*to*", the lack tense on most subordinated verbs, and so on. Such details must come from somewhere other than the message, and some aspect of the realization process must supply the representations and control resources to insure that they are used correctly. One plausible desiderata on the design of grammars for generation is thus that they coordinate the expression of the purely linguistic relations of the language while minimizing their intrusion into the process of realizing the information and rhetorical goals given by the message.

6.1 Augmented transition networks in generation

Almost immediately after Woods' defined the augmented transition network (ATN) formalism for language parsing, it was applied to generation by Jonathan Slocum and Bob Simmons (1972). The ATN was a natural approach to take to the representation of a grammar at that time, and, for the (admittedly quite simple) messages that Simmons and Slocum were working with, satisfied the condition of supporting the production of linguistic relations without adding any complications to the realization process.

ATNs for generation (see also Shapiro, 1982) have essentially the same organization as their parsing counterparts, with the obvious exception that where a parser will scan a word from the input text as an action on an arc, these generators will produce one. Their network organization mimics the recursive and sequential phrasal consitutent patterns of the language: there is a network for clauses, one for noun phrases, etc.; the clause network first looks for conditions in the message affecting whether there is an initial complement (as in questions or relatives), then determines and produces the subject, the verb and its auxiliaries, the verbal complements, and so on in left to right surface order. Grammatical function words like "*to*" or "*the*" are produced by arcs that are taken automatically once the grammatical character of the phrase has been determined. Subordinate clausal morphology, the bane of direct replacement designs, is supplied by optional arcs under the control of registers that are set by push operations in the dominating network where the grammatical function of the clause is defined. These grammatical activities take place transparently as a by-product of the motion through the networks induced by the examination of the message. They are easy to state in the notation of the grammar since they are just another part of the ATN's procedural representation of the possible surface structures that sentences in the language can have: a different structure for each coherent pathway through the networks.

The possibilities for control in an ATN are exhaustively defined by these pathways. Determination of which alternative path is taken is done by tests on the arcs, which apply predicates to the message structure. Depending on the state of the message (as defined by the tests) a particular sequence of arcs is traversed, and thus a particular text produced with the word choice and ordering given by the associated arc actions. This control technique is the diametric opposite of the more common "message as program" design (e.g. direct replacement), where initiative rests in the structure of the message rather than of the grammar. Its benefit is to simplify the statement of grammatical constraints (stated tacitly by the restrictions on possible next states imposed by the network structure), since they fall out directly from the extensional description of surface forms that the ATN supplies.

The ATN control protocol has two deficits: it wastes tests; and its tests must be very closely tied to the conceptual represention of the particular underlying program it is working with. The waste is simply that most tests will turn out to be false: only one of the arcs coming out of any node is going to be the correct one to take. With a reasonably high branching factor (reflecting the large number of alternative surface sequences that a language like English provides), it is unlikely that the correct arc will happen to always have been ordered early in the arc set. The tying of predicate definitions to the particulars of the underlying program's conceptualizations is a consequence of the ATN design having placed responsibility for determining the surface form of the text (i.e. the network path) within the grammar, rather than as a separate part of the realization component. To move an ATN-based generator to a different underlying program than the one for which it was originally developed, one with a different representional formalism or even just a different world view, will force changes in most all of the predicate definitions. (To this author's knowledge no ATN generator has ever been moved between programs; it is easy to see why.)

One final point about the ATN as a generation formalism is the unusual fact, for an ATN, that they do not ever backup (Stuart Shapiro, pers. comm. 1979). They make no use of the nondeterminism that the ATN formalism provides for, i.e. they have no need for the indeterminancy of categorization that parsers need when scanning the words of a text and predicting, top down, what structure it has. This should not be unexpected--generation is a matter of planning rather than analysis; its information flow is the addition of constraints and the progressive refinement of a completely identified intention and content, not the tentative increase in the specificity of identification that characterizes parsing.

6.2 Systemic grammars

The ATN approach to generation places control of the generation process squarely within its grammar. This drastically simplifies the programming effort required to define the procedures for constructing grammatically complex linguistic forms over what a direct replacement approach would have required, and makes it practical to use them in generation systems. However when viewed as planners, ATN generation systems suffer from too tight a coupling between perception and action: no sooner is the conceptual source of, e.g., the subject noun phrase found in the message than it is passed through the NP network of the grammar and realized as a stream of words. This hinders efforts to plan in progressive stages, determining and propagating sets of constraints or partial descriptions of what is to be generated, which is a more controllable technique than working directly in terms of surface forms that are constructed in detail as soon as they are selected.

By contrast, the linguistic school known as "systemic grammar" views surface forms as the consequences of selecting a set of abstract, functional features. The interpolation of an intermediate, abstract level of representation between the message and the output text allows the specification of the text to accumulate gradually, giving constraints an opportunity to propagate and influence later decisions, as is customary in modern planning designs. As this is not general planning, the constraints are quite specific: they are features that describe the possible functions that texts can serve, an apt theoretical vocabulary when decisions are based on how well alternatives fit intentional and communicative goals.

It is this concern for directly representing the choices a language provides and their functional consequences that has given systemic grammar its special significance in generation research. It is the only well known linguistic formalism with this special focus, the only one that makes the specification of possible choices the core of its notation. The founder of the systemic tradition, Michael Halliday, believes that language function has been neglected in linguistic theory in favor of the classification of language structures. While structure (e.g. the patterns of word and phrasal categories, clause and adjunct types etc.) is more accessible to study, explanatory theories of the structure that a language can have are arguably of little consequence without companion theories of the functions the structures can serve. Without theories that attempt to capture the criteria that govern when particular structures are used--the situational and intentional factors that contribute to the decisions made during generation--one cannot be sure that the modularity proposed in structural theories is in fact well founded.

[[[place figure two about here]]]

As in an ATN, grammaticality of the text is insured by forcing the generation process to stay within predefined paths in the systemic network; however here the paths do not define surface phrasal sequences but dependencies between abstract text characteristics. Individual features are selected independently, subject only to the dependencies defined by "systems" that group the features into networks of disjoint classes (see figure). Choice systems are given either as conjuncts (leading curled brace): where one choice must be made from each of the systems named on the right, or as disjuncts (leading square brace): where only one of the alternative features listed may be selected. The selection of a feature opens the system that it names (n.b. this feature will be the label on the horizontal line projecting leftwards from the system), which means that a choice from that system must now be made. Specific feature choices early, leftward, in the network, force later choices to be made among the features systems to the right that are dependent on them (as indicated by the connecting lines), while making other sets of choices irrelevant.

Two important generation systems have been based on systemic grammar, Anthony Davey's PROTEUS (1974/1978), and William Mann and Christian Mattheissen's NIGEL (1982, 1985). NIGEL is the largest systemic grammar in the world and very likely one of the largest machine grammars of any sort. Besides the quite important contribution simply of articulating a systemic grammar so thoroughly, Mann and Mattheiessen have developed a technique for formalizing the usage criteria that govern the choices the grammar defines, which they term "inquiry semantics" (Mann, 1983). For each choice system in the grammar a set of criterial predicates (tests) known as a "chooser" is defined. These tests are given as functions from the internal state of the planner and underlying program to one of the features in the system the chooser is associated with. The generation process consists of starting at the leftmost entry system of the nextwork and evaluating sucessive choosers to determine the path through the network (i.e. the feature set) that best captures the speaker's intentions. Davey's PROTEUS employed a very similar control protocol.

The systems and dependencies of the grammar are arranged into connected networks whose modularity matches the large-scale structural options of the language. There is a network for clauses, one for noun phrases, adjective phrases, and adverbials--all of the major phrasal categories of English. (In Davey's published grammar, the clause network includes alternatives for verb structure and prepositional complements, and incorporates the differences between major, subordinate, and relative clauses). Sentences are constructed recursively from the large structural elements downward: A sweep through the choice systems of the clause level network selects a specific subset of the concrete functional features given at the network's rightmost edge. These features collectively specify the structural form of the clause, and impose functional constraints on the conceptual elements it embeds. The "realization" of this feature set (done by an independent module) yields a sequence of grammatically and functionally marked words and embedded elements not unlike the lists found in the intermediate stages of a direct replacement generator. The functional analysis then applies recursively to the now constrained elements, making the decisions required by the connectivity of the systems inorder to functionally specify the form of the noun phrases, the subordiate clauses, and so on that the elements will become. The process of specifying functions and realizing them as form continues until all the conceptual elements referred to in the message have been realized as words, whereupon the accumulated word string is printed out. Careful attention to how much functionality is encoded at each sucessive system of alternative features in the network allows dependencies to propagate consistently (n.b. the presence of disjunctive systems means that several loci of decision-making will be operating simultaniously in a network). The result is that all decisions can be made indelibly, i.e. without ever needing to backup to an earlier system to remake the decision, and the generation process as a whole can operate in time proportional to the depth of its networks.

7. STATING THE CONSTRAINTS OF THE GRAMMAR INDEPENDENTLY

In their overall design, the ATN and systemic grammar approaches to generation differ only in the kind of information they encode in their networks: ATNs specify choices between alternative surface structure patterns; systemic grammars specify choices between alternative sets of abstract functional features. Both use their networks as their control structure, making the definition of the grammar and the definition of the actions the generator can take inseparable. Both presume that it is simpler to state the realization rules for underlying program objects by having tests in the grammar look back to the message (which is most likely just to be the unsorted state of the underlying program at the time the generator is called), than to organize the rules around the message objects and have them project forward to linguistic structures. Consequently both designs impose on their underlying programs the specific world view--set of conceptualizations--that is tacit in their grammars.[6] This is not necessarily bad: the hypothesis that one's language shapes one's thought has a long and serious history in studies of human linguistics (in an early form as the Shapir-Whorf hypothesis). It does however carry the pragmatic implication that compatibility at the conceptual level must be carefully considered when either approach to generation is used in a man-machine interface.

[6] Of course the causal dependency may go the other way: the conceptual model of the underlying program may have been worked out first and the tests of the grammar then made to conform to it; the rigidity of the final combination will be the same. Given that today the linguist is usually considerably ahead of the domain specialist in the delicacy of the conceptual discriminations that he is interested in studying, the usual dependencies will be from grammar to program.

World view and grammatical model are decoupled in other modern designs, designs which, as it happens, are also message driven rather than controlled from their grammars. We will look first at functional unification grammars, where a general purpose, "weak method" for control has been adopted, and then at the description-directed approach, where control is vested in the message and especially also in a model of the text's syntactic surface structure acting as an intermediate representation. In both cases a key question is how the design insures that all of the grammatical dependencies between decisions about form will be respected when this is no longer automatically guarenteed by the control structure.

7.1 Functional unification grammars

General principles of parsimony suggest that a grammatical representation that can be employed both for generation and understanding should be preferred to one that can not. The grammars of theoretical linguists are not specific to either direction, but by the same token they are also not grammars for processing: they capture only the abstract capacities and interior relations of a language, and not its relationships to the uses that a speaker might put it to or how it would enter causally into the psychological processes by which generation, comprehension, or learning are accomplished in specific situations. The models of language use that have been developed are almost without exception strongly directional: one cannot take a typical grammar for comprehension and "reverse" it to supply a grammar for generation. Functional unification grammar (FUG), a representation developed by Martin Kay (1979, 1984), is specifically put forward as a reversible model, where the rules of the grammar may be employed in either direction without changing the way that they are represented. The best known syntactic-level FUGs are the TELEGRAM grammar developed by Doug Appelt (1985) for his generator, and the realization component developed by Steve Bossie (1981) for use with the text planning system of Kathy McKeown (1985).

The figure shows a fragment of a FUG taken from Appelt's grammar (1985, pg.108). It defines the possible constituent roles for the components of noun phrases. The enclosing brackets define systems of features and values. Square brackets define conjunctive sets: a noun phrase must specify a value for each of their features. Curly brackets define disjunctive sets: the noun phrase must meet any one of the listed feature-values.

[[[insert Figure Three about here]]]

The structure of an FUG--sets of equations that define feature/value pairs--suggests no immediate procedural interpretation beyond the "outer versus inner" segmentation of the hierarchical groupings; this lack of procedure in the representation's interpretation is a requirement of any potentially bi-directional formalism. The computational role of the FUG is as a set of constraints that any independently defined text construction procedure must meet. This makes it not a source of control but a filter on the control options of some other source or sources: in this instance the weak method known as nondeterministic unification. We will look at Appelt's use of a FUG as an example.

The focus of Appelt's work on generation is on the planning process, and relatively few details of how his planner initiates and regulates the realization process are available. However it seems fair to characterize those aspects of his design as message-directed, in the sense that it is objects and relations selected by the planner that initate realization, under the direction of concept-specific specialist programs as in direct replacement designs.[7] The specialists invoked by the text planner begin the process by building skeletal descriptions of

[7] Appelt's KAMP planner feeds the linguistic levels of processing incrementally, specifying additional conceptual information for realization, as well as where it should appear, whenever the decision to include that information is made during the planning process. Given this, it makes no sense to say that there is ever an explicit "message" in the sense of a structure planned as a unit and then realized without further intervention by the planner.

the constituents that the text is to include, for example that there is to be a clause with a certain main verb and two NPs with certain lexical heads. (The referents of these phrases will be indicated by pointers back into the underlying program model that can be referred to as needed.) The FUG is then used to extend such skeletal descriptions with the additional linguistic features that are required for the text to be grammatical.

The extension of the specialists' descriptions is done by merging them with the grammar, constituent by constituent. This works because the grammar is effectively a maximal description of the possible constituents, functional relationships, and constraints of the language. It defines the linguistic resources that the specialists can draw on, and specifies grammatical details that no conceptual-level planner would be concerned about. The merger of the initial, partial description of what the specialists want said with the grammar's description of what the language permits is so organized that only extensions consistent with the grammar will occur. The grammar supplies the constraints that proposed functional descriptions must meet, and, through its disjunctive feature sets, supplies the choice sets from which many of the extensions will come. The merger is governed by the semantics of the unification operator: features from the grammar may be added if they contradict none that are already present or if they subsume them.

The specialist's partial descriptions will prompt the addition of the features from the grammar that are tied to those in the description by functional links (see Kay 1984 for details). This instantiation of previously unspecified features then has a ripple effect throughout the whole system: Other features that are dependent on a just instantiated feature force further unifications cyclically until a grammatically complete description has been formed. During the process feature values in the description will force selections among the disjunctive specifications in the grammar. For example specifying a verb will force the grammatical subcategorization feature to take a compatible value, which in turn will impose constraints on the selections available to the noun phrases that the subcategorization pattern governs. The completed description will amount to a rooted tree of feature specifications as defined by the "pat" (pattern) feature that dictates the sequential order of constituents at each level. The actual production of the text is performed by scanning this tree and reading out the words in the lexical features of each constituent.

From the point of view of grammar development, FUGs are a satisfying treatment because they allow one to state the facts of the language compactly, i.e. interactions between statements need not be explicitly spelled out in the notation since they will come about automatically through the action of unification. However, from a processing point of view, this convenience comes with a price, since one must be willing to live with the tacit control structure that does the "behind the scenes" merging of descriptions and grammar. In the present implementations, this is nondeterministic unification. If efficiency of execution is not relevant then this of course is no problem, however there are indications (Graeme Richie, 1986) that the computational properties of FUGs make general operations over them NP-complete. Since language is a very specific phenomena, one would hope that equally specific, more computationally efficient processing schemes would suffice, rather than having to rely on such a general, and therefore powerful, mechanism. The next approach to be discussed, description-directed generation was designed with such efficiency concerns in mind.

Before leaving this section, it is important to point out that recent work by Tony Patten (1985) uses a systemic grammar in very much the same way that FUGs are used. In Patten's approach, operations at a semantic level (of the same kind as performed elsewhere by planning level specialists) specify a set of output features within the systemic grammar, i.e. features at the rightmost edge of the network. These are the equivalent of the initial functional description that drives a FUG. A backwards and then forwards chaining sweep through the system network determines what additional linguistic features must be added to the specification for a grammatical text to result, just as unification does with the FUG. This points out that systemic grammars can be viewed just as reference constraint sets and need not dictate the total control flow as is usually done.

8. Multi-Level, Description Directed Generation

The principal deficit of the direct replacement approach is its difficulties with grammar, i.e. the awkwardness of maintaining an adequate representation of the grammatical context, or of carrying out grammatically mediated text-level actions such as producing the correct syntactic form for an embedded clause. In other respects, however, the message-directed control flow that drives direct replacement has a great deal to recommend it. Compared with grammar-directed control schemes, message-directed control is more efficient, since every action will contribute to the eventual production of the text. Message-directed control also gives a planner a very clear semantic basis for its communication to the realization component, since the message can be viewed simply as a set of instructions to accomplish specific goals. The question then becomes: is there a way of elaborating the basic, message-directed framework so as to overcome the deficits that plague direct replacement approaches while still keeping the computational properties that have made it attractive?

A number of generation researchers have independently choosen the same solution: to interpose a level of explicitly linguistic representation between the message and the words of the text (McDonald, 1975, 1984; Kempen and Hoenkamp, 1982; Jacobs, 1985; Swartout, 1984). They believe that employing a syntactic description of the text under construction is the most effective means of introducing grammatical information and constraints into the realization process, in particular, that it is a better locus for grammatical processing than a separately stated, active grammar.

The specifics of their individual treatments differ, but a common thread is clearly identifiable: Realization is organized as choices made by specialists, where the form of the choice--the output of the specialist--is a linguistic representation of what is to be said, i.e. a structural annotation of the syntactic relations that govern the words to be said (and embedded conceptual elements), rather than just a list of words. These representations are phrase structures of one or another sort--hierarchies of nodes and constituents--of essentially the same kind that a theoretical linguist would use. They employ functional terms like "subject" and "focus", and are most aptly characterized as a kind of "surface structure" in the generative linguist's sense, e.g. they undergo no derivation, and are a proper and complete description of the syntactic properties of the text that is produced.

It will be convenient to restrict the present discussion to only one examplar of this approach; taking advantage of an author's prerogative, I will describe my own (c.f. McDonald, 1984; McDonald & Pustejovsky, 1985b; McDonald, Pustejovsky & Vaughan, 1986). As it is the historical outgrowth of a direct replacement system,[8] it will be useful to organize the discussion in terms of how it extends that approach and addresses its deficits. This will be folded into the standard description of how it deals with the three general concerns one should have in examining a generation system: how it organizes its knowledge of grammar; what its control structure is; and what its approach to realization is.

Referring to our approach as "multi-level, description-directed generation" emphasizes specific features of its architecture and control protocols that we consider important; it is, however, too large a phrase to use conveniently. The name of the computer program that implements the design, MUMBLE (McDonald, 1977,1983), will serve as a compact, agentive reference. Characterizing MUMBLE as multi-level draws attention to the fact that it carries out operations over three explicitly represented levels of representation simultaneously: message, surface structure, and word stream. Description-directed is the name we have given to its control protocol, which is a specialization of the common programming technique known as data-directed control. Under this protocol, the data in

[8] This author's interest in natural language generation began in 1971 while he was working on extentions to the grammar and parser in Winograd's SHRDLU program. As already discussed, SHRDLU employed a classic direct replacement technique for its generation. It was observations of the shortcomings of that design that were the original motivation for the research. The influences of systemic grammar and data-directed programming style also stem from that time.

the representations at the three levels is interpreted directly as instructions to the virtual machine that constitutes the generator proper. Since each of these representational structures is also a valid description of the text at its own level of abstraction and theoretical vocabulary, this characterization of the protocol emphasizes the fact that the particulars of how the person developing the messages or syntactic structures chooses to design them has immediate consequences for the generator's performance (McDonald, 1984). The feedback that this gives a developer has proven to be invaluable in refining the notations and their computational interpretations in all parts of the system.

MUMBLE's virtual machine is the embodyment of our computational theory of generation. It consists of three interleaved processes that manage and carry out the transitions between the representational layers. (1) Phrase structure execution interprets the surface structure, maintaining an environment that defines the grammatical constraints active at any moment, and producing the word stream as its incremental output. (2) Attachment interprets the message, transferring its component units to positions within the surface structure according to the functional relationships between them and their role in the message. (3) Realization takes the individual elements of the message into surface structure phrases by selecting from linguistically motivated classes of parameterized alternative forms. A minor fourth process, operating over the word stream, morphologically specalizes individual words to suit their syntactic and orthographic contexts (e.g. the article"*a*" going to "*an*" before vowels); later versions of MUMBLE that produce speech should be much more active at this level.

Thus, as seen by the developer of a text planner that would pass messages to MUMBLE for it to produce texts from, the virtual machine appears as a very high level, task-specific language, with its own operators and intermediate representations. To a lesser extent this is true also for the linguist writing generation-oriented grammars for MUMBLE to execute, since the virtual machine includes no presumptions as to what specific syntactic categories, functional relations, or syntactic constructions the natural language includes. Instead it supplies a notation for defining them in terms of primitive notions including the dominates and proceeds relations of phrase structure, bound thematic relations, configural regularities such as "head" or "complement" from X-bar theory, and the tree combination rules of Tree Adjoining Grammars (Kroch & Joshi, 1985).

As a message-directed design, MUMBLE is best discussed by reference to a concrete example message, situation, and resulting output text. To miminize the distraction that introducing an actual underlying program from one of our generation projects would entail, a relatively obvious excerpt from a message will have to suffice. Figure Four shows a generated output paragraph describing a legal case from the UMass Counselor Project (McDonald & Pustejovsky 1986). The structure below it is the message responsible for its second sentence, which details the events that were relevant to the court's decision. Using this example, we will look at MUMBLE's knowledge of grammar: how it is manifest, and how it has its effects, interleaving discussion of realization and control at convenient places.

"In the Telex case, Telex was sued by IBM for misappropriating trade secrets about its product Merlin. One of the managers of the Merlin development project, Clemens, left IBM to work for Telex, where he helped to develop Telex's competing product, the 6830. The key fact in the case was that Clemens brought a copy of the source code with him when he switched jobs. The court held for IBM."

```
(temporal-sequence
    (left-to-work-for   (#<role #<project-manager Merlin>>
                         #<Clemens>)
                        (named-company  #<IBM>)
                        (named-company  #<Telex>))
    (helped-to-develop   (named-person #<Clements>)
                         (#<kind product>
                          #<competition-by #<Telex>>
                          #<name "6830">) ))
```

FIGURE FOUR

As previously discussed, one of the concomitant features of a message-directed approach is that items[9] directly from the underlying program are part of the messages. (These are indicated here by enclosing angle brackets, #<...>.) Once in a message, such items become instructions to the generator, and as such need interpretations, i.e. associated functions from the item, and the linguistic and pragmatic environment, to the surface specification of some text or text fragment. However, considered in terms of the space of texts that might realize them, real program objects are large and vague as present day programmers tend to use them: they stand in many different relationships to other objects and to the underlying program's state, and consequently can have many different interpretations depending on the context and the speaker's intent.

We take it to be part of the job of a text planner to choose among these relationships and to indicate in the message the perspective from which an object is to be viewed. The perspective on the first occurance of Clemens, for example, is indicated to be his role as (former) manager of the Merlin project. Adopting a specific perspective often amounts to selecting a specific wording (often just of the lexical head, e.g. "*manager*"; but also entire conventional phrases such as "*leave* <employer1> *to work for* <employer2>"). These examples indicate that many of the terms in a message are surface lexical relations (e.g. "*helped to develop*") rather than a more abstract conceptual vocabulary; this has the deliberate corollary that syntactic realization will usually occur <u>after</u> key words have been chosen. The text planner must therefore understand a good deal about how alternative word choices cover the semantic fields of the situation it is trying to communicate, and what emphasis and what presupposed inferencing by the audience a given choice of wording will convey. This appears to us to be a choice that is best made at a conceptual level (i.e. during message construction), since it does not depend in any crutial way on the details of the grammatical environment, the arguments of Danlos (1984) notwithstanding (cf. McDonald et al. 1986).

Even though the key lexical choices for an item will have occurred before it has been syntactically realized, these message-level lexical decisions can draw on the grammatical context in which the text for it is going to occur. In particular, grammatical constraints imposed by the syntactic relations in which the text will stand will filter out grammatically

[9] The word "item", and at other times the word "object", is intended as a general term that denotes representational data structures in an underlying program without regard to the kind of real world entity that they model: individuals, kinds, relations, constraints, attributes, states, actions, events, etc.

inconsistent possibilities from the planner's choice set.[10] This is possible because the realization of messages is hierarchical, following the message's compositional structure top down, i.e. the message is interpreted much as a conventional program would be. The surface syntactic realization of the higher, dominating conceptual elements of the message is thus available to define and constrain the interpretations (i.e. linguistic realizations) of the lower, more embedded elements. This protocol for "evaluation" of arguments is known as normal order, and is in direct contrast with the previously discussed applicative order protocol used in most direct replacement designs.

The perspective that the text planner chooses to impose on an item from the underlying program is represented at the message-level by designating the realization class to be used for it. Realization classes are MUMBLE's equivalent of the "specialist programs" in direct replacement. They are linguistic entities rather than conceptual, and are developed by the designer of the grammar using control and data structures defined in the virtual machine. New underlying programs are interfaced to MUMBLE by developing a (possibly very minimal) text planner and assigning program items (or item types) to pre-defined realization classes. A relatively self-contained example of a class, "locative-relation", developed originally for use with Jeff Conklin's program for describing pictures of house scenes (see Conklin, 1984) is shown below:

```
(define-realization-class    Locative-relation
  :parameters   (relation  arg1  arg2)
  :choices
       ( (Arg1-is-Relation-Arg2)
         "The driveway is next to the house"
         clause    focus(arg1) )
       ( (Arg2-has-Arg1-Relation-Arg2)
         "The house has a driveway in front of it"
         clause    focus(arg2) )
       ( (There-is-a-Arg1-Relation-Arg2)
         "There is a driveway next to the house"
         root-clause    shifts-focus-to(arg1)  )
       ( (Relation-Arg2-is-Arg1)
         "Next to the house is a driveway"
         root-clause    shifts-focus-to(arg1)
         final-position(arg1)  )
       ( (with-Arg1-Relation-Arg2)
         "...with a driveway next to it"
         prepp    modifier-to(arg1)  ))
```

The choices grouped together in a realization class will all be effective in communicating the conceptual item assigned to the class, but each will be appropriate for a different context. This context-sensitivity is indicated in the annotation accompanying the choice, for example "focus", which will dictate the grammatical cases and surface order

[10] This filtering is automatic if the relevant parts of the text planner are implemented using the same abstract control device as MUMBLE uses for its own decisions, i.e. parameterized, pre-computed annotated choice sets of the sort employed for realization classes (see text). The descriptions of the lingustic character and potential of the choices that the annotation provides are the basis for filtering out incompatible choices on grammatical grounds, just as occurs at the syntactic level in selections within a realization class.

This technique is proving convenient in our own work with some simple text planners; however we can see a point where the requirement that the full set of alternatives be pre-computed may be unnecessarily limiting or possibly psychologically unrealistic, in which case an alternative design, presumably involving dynamic construction of the choices, will be needed and an alternative means of imposing the grammatical constraints will have to be found. For a discussion of another planning-level control paradigm that has been used with MUMBLE, see Conklin (1984) or McDonald & Conklin (1983).

given to the arguments, or the functional role "modifier-to", which will lead to realization as a postnominal prepositional phrase. These annotating characteristics indicate the contexts in which a choice can be used. They act both as passive descriptions of the choice that are examined by other routines, and as active test predicates that sample and define the pragmatic situation in the text planner or underlying program. Such terms are the basis of MUMBLE's model of language use--the effects that can be achieved by using a particular linguistic form; as such they play the same kind of role as the "choosers" or the controlling functional features in a systemic grammar like Mann's NIGEL.

The surface structure level, the source of grammatical constraints on realization, is assembled top down as the consequence of the interpretation and realization of the items in the message. In the example message (repeated below), the topmost item is a "sequence" of two steps, each of which is a lexicalized relation over several program objects on which a particular perspective has been imposed.

```
(temporal-sequence
    (left-to-work-for  (#<role #<project-manager Merlin>>
                        #<Clemens>)
                       (named-company  #<IBM>)
                       (named-company  #<Telex>))
    (helped-to-develop  (named-person #<Clements>)
                        (#<kind product>
                         #<competition-by #<Telex>>
                         #<name "6830">) ))
```

One of the goals of a multi-level approach is to distribute the text construction effort and knowledge throughout the system so that no level is forced to do more of the work than it has the natural capacity for. Thus for example in the interpretation of the first item the message, `temporal-sequence`, MUMBLE is careful to avoid taking steps that would exceed the intent of the planner's instruction by being overly specific linguistically: As a message-level instruction, `temporal-sequence` says nothing about whether the items it dominates should appear as two sentences or one; it says simply that they occured after one another in time and that their realizations should indicate this. Since there is no special emphasis marked, this can be done by having them appear in the text in the order that they have in the message. The decision about their sentential texture is postponed until a linguistic context is available and the decision can be made on an informed basis.

This delay is achieved by having the Attachment process, which moves items from the message to the surface structure according to their functional roles, wait to position the second item of the sequence until the first has been realized. Only the first item will be moved into the surface structure initially, and it will appear as the contents of the second sentence as shown below in Figure Five. Note that a message item is not realized until it has a position, and then not until all of the items above it and to its left have been realized and the item has been reached by the Phrase Structure Execution process that is traversing the surface structure tree and coordinating all of these activities. By enforcing this discipline one is sure that all the grammatical constraints that could affect an item's realization will have been determined before the realization occurs, and consequently the virtual machine does not need to make provisions for changing an item's realization after it is finished.

Considered as a function, a realization class such as "Left-to-work-for" specifies the surface form of a grammatically coherent text fragment, which is instantiated when the class is executed and a specific version of that phrase selected. Given its lexical specificity, such a class is obviously not primitive. It is derived by sucessive specializations of two, linguistically primitive subcategorization frames: one built around the verb class that includes "*leave*" (shown below) and the other around the class containing "*work for* ". The specialization is done by a definition-time currying operation wherein arguments to the subcategorization frames are bound to constants (e.g. the verb "*leave*"), producing new realization classes of reduced arity. On its face, a class built around variants on the phrase "<employee> *leaves* <company1> *to work for* <company2>" is more appropriate to a semantic grammar (cf. Burton & Brown, 1977) than to a conventional syntactic phrase structure grammar. This choice of linguistic modularity does however reflect the actual conceptual modularity of the underlying program that drives the example,[11] and we believe this is an important benefit methodologically.

```
(define-phrase  subject-verb-locative  (subj vb loc)
     :specification  (clause
                          subject  subj
                          predicate  (vp
                                          verb  vb
                                          locative-complement  loc )) )
```

Comparing MUMBLE's organization of grammatical knowledge with that of the two grammar-directed approaches that have been discussed, we see that it resembles an ATN somewhat and a NIGEL-style systemic grammar hardly at all. ATN designs are based on procedurally encoded surface structures, which are executed directly; MUMBLE represents surface structure explicitly and has it interpreted. ATNs select the surface form to be used via a recursive, phrase by phrase, topdown and left to right consideration of the total set of forms the grammar makes available (i.e. alternative arc sequences), and queries the state of the underlying program to see which form is most appropriate. MUMBLE also preceeds recursively, topdown and left to right, but the recursion is on the structure of an explicitly represented message. Conceptual items or item types, through the the realization classes that the planner associates with them, control the selection and instantiation of the appropriate surface forms directly.

MUMBLE "packages" linguistic relations into constituent phrases; it does not provide an unbundled, feature-based representation of them as a systemic grammar does. It cannot, for example, reason about tense or thematic focus apart from a surface structure configuration that exhibits them. This design choice is deliberate, and reflects what we take to be a strong hypothesis about the character of linguistic knowledge. This hypothesis is roughly that the space of valid feature configurations (to use systemic terms) is smaller, less arbitrary, and more structured than a feature-heap notation can express (see McDonald et al. 1986 for details). Since our notation for surface structure incorporates functional annotations as well as categorical, and especially since it is only one of three representational levels operated over in coordination, we believe that organizing linguistic reasoning in terms of packaged, natural sets of relations will provide a great deal of leverage in research on text planning and computational theories of language use and communicative intention.

Nowhere in MUMBLE is there a distinct grammar in the sense of a set of rules for deriving linguistic forms from primitive features. Rather MUMBLE manipulates a collection of predefined linguistic objects--the minimal surface phrases of the language and the composite phrases derived from them. The phrases are grouped into the realization classes--the projected linguistic images of different conceptual types and perspectives. When selected and instantiated to form the surface structure they take on an active role (through

[11] As it happens, Leave-to-work-at is a primitive conceptual relation in the legal reasoning system that serves here as the underlying program (Rissland & Ashley, submitted). The causal model that the phrase evokes in a person, i.e. that working for the new company is the reason why the employee is leaving (cf. "*John washed his car to impress his girlfriend*") is encapsulated in this relation, and suppresses the causal model from consideration by the legal reasoner's rules. This encapsulation is deliberate. Reasoning systems should function at the conceptual level best suited to the task. This does however imply that some component of the natural language interface must now bridge the conceptual ground between the internal model and the lexical options of the language; see Pustejovsky (this volume) for a discussion of how this may be done.

interpretation by the three processes), defining the order of further actions by the generator, defining the contraints on the realization of the embedded items from the message now at some of its leaf positions, and defining the points where it may be extended through further attachments from the message level. Figure Six shows a snapshot of the surface structure for the first part of the text in the example, and can illustrate these points. At the moment of this snapshot, the Phrase Structure Execution process has traversed the structure up to the item `#<telex>` and produced the text shown; its next action will be to have that item realized, whereupon the realizing phrase (an NP like the one for `#<IBM>`) will replace `#<telex>` in the surface structure and the process will traverse it and move on (see figure two).

The first thing to consider is the differences in the details of this surface structure representation compared with the more conventional trees used by generative grammarians. Two of these are significant in this discussion. The first is the presence of functional annotations over each of the constituents (indicated by labels inside square brackets). Terms like "subject" or "prep-complement" are used principally to summarize the grammatical relations that the constituents are in by warrant of their configurational positions, which makes these labels the source of most of the grammatical constraints on message item realizations. The functional annotations also play a role in the dynamic production of the word stream: Here this includes providing access to the subject when the morphological process needs to determine the person/number agreement for tensed verbs, and supplying grammatical function words like "*of*" or the infinitive marker "*to*" directly into the word stream.[12]

Formally the representation is not a tree but a sequential stream (as indicated by the arrows): a stream of annotated positions that are interpreted, in order, as instructions to the Phrase Structure Execution process. The grammar writer defines the interpretation an annotating label is to have, e.g. specifying control of morphological effects or function words, constraints to be imposed on realizations, or establishing salient reference positions (like the subject). Various useful technical details are expedited by defining the surface structure as a stream rather than a tree (see McDonald & Pustejovsky 1985b). The stream design provides a clean technical basis for the work of the Attachment process, which extends the surface structure through the addition of successive items from the message. The extensions are integrated into the active grammatical environment by breaking inter-position links in the stream and kniting in the new items along with any additional covering syntactic nodes or functional constituent positions needed to correctly characterize the linguistic relationship of the new material to the old.

In the present example, the second item of the message's temporal sequence item, the lexicalized relation "helped-to-develop", remains unattached--its position in the surface structure unestablished--*until* enough linguistic context has been established that a reasonable decision can be made about stylistic matters, e.g. whether the item should appear as an extension of the first item's sentence or start its own. Since the functional constraints on a temporal sequence's realization prohibit embedding the second item anywhere within the first, the only legal "attachment points" for it (i.e. links it could be knit in at) are on the trailing edge of the first item's sentence or as a following sentence. In

[12] Introducing the closed class words that indicate syntactic function into the text as an active consequence of traversing the corresponding part of the surface structure tree, rather than having them first appear in constituent positions at the tree's leaves, is an experimentally motivated design decision. It is intended to explore the computational consequences of employing grammars that distinquish the sources of closed and open class words: positing that the open class words have a conceptual source and the closed class, "function" words a purely syntactic source. The two word classes are distinguished psycholinguistically, e.g. they have very different behaviors in exchange errors (see Garrett 1975); if this empirical difference can be given a successful computational account, then that account can serve to anchor other aspects of the grammar's design and eventually lead to psycholinguistic predictions derived from the consequences of the computational design (McDonald 1984).

terms of our theory of generation, attachment points are grammatical properties of phrasal configurations: places where the existing surface structure may be extended by splicing in "auxiliary" phrases (i.e. realizations of message items), for example adding an initial adjunct phrase to a clause or embedding the NP headed by "manager" inside the selector "one of". Every phrasal pattern (as indicated by the annotating labels) has specific places where it can be extended and still be a grammatically valid surface structure; the grammatical theory of such extensions is developed in studies of Tree Adjoining Grammars (Kroch & Joshi 1985).

What attachment points exist is a matter determined by the grammatical facts of the language; which points are actually used in a given situation is a matter of stylistic convention (see McDonald & Pustejovsky 1985a). In this case there is a very natural, compactly realized relationship between the first and second temporal events: the final item in the realization of the first event, the Telex company, happens to be where the second event occurred. As neither clause is particularly complex syntactically, the attachment point that extends the final NP of the first event with a relative clause is taken and the second event knit into the surface structure there, to be realized when that position is reached in the stream.

9. Relating Modern Generation Approaches to MT

Like the AI generation researchers themselves, the principal problem for MT researchers who want to take advantage of any of the modern work on generation is what to start from. They cannot just take a generator design "off the shelf" and use it without modification--MT systems do not have the information about intention and conceptual perspective that the AI designs presume is available to guide decisions. Conceivably this information could be deduced as the source text is parsed, but only at the cost of including a deep conceptual model of the subject matter of the text, which is unrealistic for practical systems.

It would however be a mistake to think that generation in MT must be limited by the depth of the source analysis. The only things that a reconstruction of the author of the source text's intent provides is the freedom to be unconstrained by the the specific way that the information in the source text was structured, i.e. by starting the generation at its first stage, the identification of the speaker's goals, one can make arbitrary choices of wording or even of the relations to use in planning how the text will be organized. One must ask though if this freedom is really necessary. Is there that much conceptual incompatibility between languages that isomorphic source and target texts cannot be accepted? The translation of literature does seem to require this, but that is a creative activity usually done by professional writers and poets; it may not be translation so much as a retelling of the story in another language and culture. If that were taken as the goal of MT, the field would have fallen prey to the superhuman human fallacy: expecting to formalize a natural ability that only exceptional people have.

But if an MT is not to start generating where an interface generator would, then where should it? The key to answering this question comes in a notion mentioned briefly at the beginning of this paper, that of "structure-fixing information". If we view a text as the set of decisions that are made during its construction,[13] then the *structure-fixing information* for that text is the information that directs those decisions. In an interface generator, the initial structure-fixing information is supplied entirely by the pragmatic state of the underlying program, supplemented eventually by the structures produced by the planner and linguistic generator as the process proceeds. In an MT system, the situation is reversed: the most readily available information is the linguistic and rhetorical structure recovered by the parser as it processes the source text; pragmatic information about the original speaker's state is either never adduced at all or at best only tentatively asserted. To utilize the parser's information it must be converted into terms a generator can use, and fed to it at the appropriate moments.

Because it has "been there before" during the parsing of the source, an MT system is able to directly contribute structure-fixing information to almost all the decision points in the generation process, especially, of course, those involving word choice and syntactic structure. To use this information to best advantage, it should be couched in a form that a powerful generator can use (i.e. one that could be used, by itself, in an interface with a conversationally adroit underlying program). This would not be direct translation of the source forms: if that sufficed then MT would be a solved problem. Rather what is required is to view the parser's output as though it had been produced by a generator--as though it were a reflection of decisions a generator had made. This requires expressing the results in a compatible terminology (e.g. parse trees, functional features, etc.), and drawing on a mapping from the outputs of the generator's decisions back to the decisions themselves. Seen from that perspective, a parser's analysis can be made to constrain the generator's decisions: only decisions consistent with those imputed to the original speaker will be allowed. The structure-fixing information produced by the parser feeds the generation process wherever it can apply to a decision of the target language that had a counterpart in the source language, while decisions unique to the target language are done on the basis of a priori principles or by employing heuristic extensions of the source language decisions as narrowing constraints.

Once the parsers in mechanical translation systems can be made sensitive to the concerns that drive the rhetorical decisions of generators, then their output texts may cease to be so mechanical in their texture. Nearly any generation system can reproduce the case frame information that standard parsers recover, but the best ones use information about the *pragmatic situation* and the speaker's intent to communicate whether that information is new, salient, conventional, etc.. When parsers can notice these perspectives in source texts and associate them with the generation decisions that could have lead to them, then we should expect that MT output should be able to be every bit as good as any text produced from an interface.

[13] What constitutes a "decision" during generation will always be relative to the type of approach. In particular, decisions must be considered against the background of the specific kinds of objects that the approach takes to be involved in the generation process: arcs (in an ATN), abstract features (in FUGs or most systemic grammars), word sequences and domain-level conceptual objects (in direct replacement), or surface structure phrases (in MUMBLE and other systems that use a linguistic intermediate representation). A generator's decisions always lead to operations over these objects (comparison, selection, omission, constraint, etc.), meaning that since these objects are all quite different in nature, the "decisions" in each of these approaches will be equally different.

10. BIBLIOGRAPHY

Appelt D. (1980) Problem Solving Applied to Language Generation, Proc. ACL-80, Philadelphia, 59-63.

_____ (1985) **Planning English Sentences**, Cambridge University Press.

Bossie S. (1981) A Tactical Component for Text Generation: Sentence Generation Using a Functional Grammar, Dept. Computer & Information Science, University of Pennsylvania, TR MS-CIS-81-5.

Bruce, B. (1975) "Generation as social action", TINLAP-1, ACM, 74-78.

Burton, R. (1976) "Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems", Bolt Beranek and Newman, Inc., TR 3453, Cambridge Mass.

Chester D. (1976) The translation of formal proofs into English, **Artificial Intelligence** 8(3), 261-278.

Clancey W. (1979) Tutoring Rules for Guiding a Case Method Dialog, **IJMMS II**, 25-49.

Clippinger, J. (1977) **Meaning and Discourse: a computer model of psychoanalytic speech and cognition**, Johns Hopkins Press.

Conklin, E. (1983) Data-driven Indelible Planning of Discouse Generation Using Salience, Ph.D. thesis, Dept. of Computer and Information Science, University of Massachusetts.

Danlos L. (1984a) Generation automatique de textes en langues naturelles, These d'Etat, Universite de Paris 7.

_____ (1984b) "Conceptual and Linguistic Decisions in Generation," Proc. COLING-84, Stanford University, July 2-6, 1984, 501-504.

Davey A. (1974) Discourse Production, Ph.D. thesis, Edinburgh University; published in 1978 by Edinburgh University Press.

Forbus K., A. Stevens (1981) Using Qualitiative Simulation to Generate Explanations, Proc. Third Annual Conf. Cognitive Science Society, Berkeley, August 19-21, 1981, 219-221.

Garrett, M. (1975) The analysis of sentence production, in Bower (ed.) **Psychology of Learning and motivation, Vol. IX.**, Academic Press, 133-177.

Goldman N. (1975) "Conceptual Generation", in Schank, R. (ed.) **Conceptual Information Processing** North-Holland/Elsevier, 289-372.

Halliday M.A.K. (1969) "Options and functions in the English clause", **Brno Studies in English** 8, 82-88, collected in Halliday & Martin (eds) **Readings in Systemic Linguistics**, Batsford Academic Press, 1981, 138-145.

Hoenkamp, E. (1983) Een Computermodel van de Spreker: Psychologische en Linguistische Aspecten, Ph.D. thesis, Dept. of Psychology, Catholic University, Nijmegen, The Netherlands.

Jacobs P. (1985) A Knowledge-Based Approach to Language Production, Berkeley CS Dept TR 86/254.

Kay M. (1979) "Functional Grammar", Proc. Fifth Annual Meeting of the Berkley Linguistics Society.

__ (1984) Functional Unification Grammar: a formalism for machine translation, proc. COLING-84, Stanford, 75-78.

Kempen G. & E. Hoenkamp (1982) "Incremental sentence generation: implications for the structure of a syntactic processor", Proc. COLING-82, Prague, July 5-10 1982, 151-156.

Kroch A., & K. Joshi (1985) "The Linguistic Relevance of Tree Adjoining Grammar", Tech. Rpt. MS-CIS-85-16, Dept. of Computer Science, University of Pennsylvania.

Kukich K. (1983) Knowledge-Based Report Generation: A Knowledge Engineering Approach to Natural Language Report Generation. Ph.D. Thesis, Information Science Dept., University of Pittsburgh.

Lenat, D., M. Prakash, M. Sheperd (1985) "CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks" in **AI Magazine** VI(4), 65-85.

Lytinen, S. (1984) "The Organization of Knowledge in a Multi-lingual Integrated Parser" Ph.D. thesis, Department of Computer Science, Yale University.

Mann W. (1982) The Anatomy of a Systemic Choice, TR/RS-82-104, Information Sciences Institute, Los Angeles.

_____ (1983) Inquiry semantics: a functional semantics of natural language, ISI TR/RS-83-8.

_____, M. Bates, B. Grosz, D. McDonald, K. McKeown, W. Swartout (1982) "Text Generation: the state of the art and literature", **JACL** 8(2), 62-69.

_____ & Matthiessen (1985) Nigel: a Systemic Grammar for Text Generation, in Freedle (ed.) **Systemic Perspectives on Discourse: Selected Theoretical Papers of the 9th Intl. Systemic Workshop**, Ablex.

_____ & J. Moore (1981) Computer generation of multi-paragraph English text, **JACL** 7(1), 17-29.

McDonald D. (1975) "A Preliminary Report on a Program for Generating Natural Language", Proc. IJCAI-75, Wm. Kaufman, 401-405.

_____ (1983) "Natural Language Generation as a Computational Problem: an introduction", in Brady & Berwick (eds) **Computational Models of Discourse**, MIT Press, 209-266.

_____ (1984) Description Directed Control: Its implications for natural language generation, in Cercone (ed), **Computational Linguistics**, Plenum Press, 403-424.

_____, J. Pustejovsky (1985a) "A computational theory of prose style for natural language generation", Proc. European Chapter of the Association for Computational Linguistics, University of Geneva, March 28-30, 1985, 86-94.

_____, _____ (1985b) TAGs as a Grammatical Formalism for Generation, Proc ACL-85, University of Chicago, 94-103.

_____, _____ (1986) "The UMass Counselor Project", **JCL Finite String Newletter**, 12(2), 139-141.

_____, _____, Vaughan M., (1987) "A measure of complexity for text planning and generation" in the **Papers from the 3d International Workshop on Language Generation** to be published by Martinus Nijhoff Press (Kluwer).

McKeown K. (1985) **Text Generation**, Cambridge University Press.

Patten T. (1985) A Problem Solving Approach to Generating Text from Systemic Grammars, Proc. European Chapter of the Association for Computational Linguistics, University of Geneva, March 28-30, 1985, 251-256.

Pustejovsky J. (1986) "An integrated theory of discourse analysis", this volume.

Richie, G. (1986) "The computational complexity of sentence generation using functional unification grammar", Proc. COLING-86, Bonn, West Germany, August 25-29, to appear.

Rissland, E. & Ashley, K. (1986) "Hypotheticals as a Heuristic Device", Proc. AAAI-86, Philadelphia, August 11-15, to appear.

Simmons R. & J. Slocum (1972) Generating English discourse from semantic networks, **CACM** 15(10), 891-905.

Shapiro S. (1982) Generalized Augmented Transition Network Grammars for Generation from Semantic Networks, **JACL** 8(1), 12-25.

Slocum, J. (1984) "Machine Translation: its History, Current Status, and Future Prospects" in Proc. COLING-84, Stanford University, July 2-6, 1984, 546-561.

Swartout W. (1981) Producing Explanations and Justifications of Expert Consulting Programs, MIT/LCS/TR-251.

_______ (1983) "The GIST English Generator", Proc. AAAI-82, Pittsburg Penn.

Tait, J. (1982) "Automatic Summarising of English Texts" Ph.D. thesis, Dept. of Computer Science, Cambridge University.

Webber, Bonnie (1971) The Case for Generation, in Papers presented at the Seminar in Mathematical Linguistics, vol. XIII, ms. Aiken Computational Laboratory, Harvard University, spring term 1971.

Wilensky R., Y. Arens, D. Chin (1984) Talking to UNIX in English: An overview of UC, **CACM**, 577-593.

Winograd T. (1972) **Understanding Natural Language**, Academic Press.

Wolf, B. (1984) "Context Dependent Planning in a Machine Tutor", Ph.D. thesis, Dept. of Computer & Information Science, University of Massachusetts.

Wolf, B. & McDonald, D. (1984) "Building a Computer Tutor: Design Issues" **IEEE Computer** 17(9), 61-73.

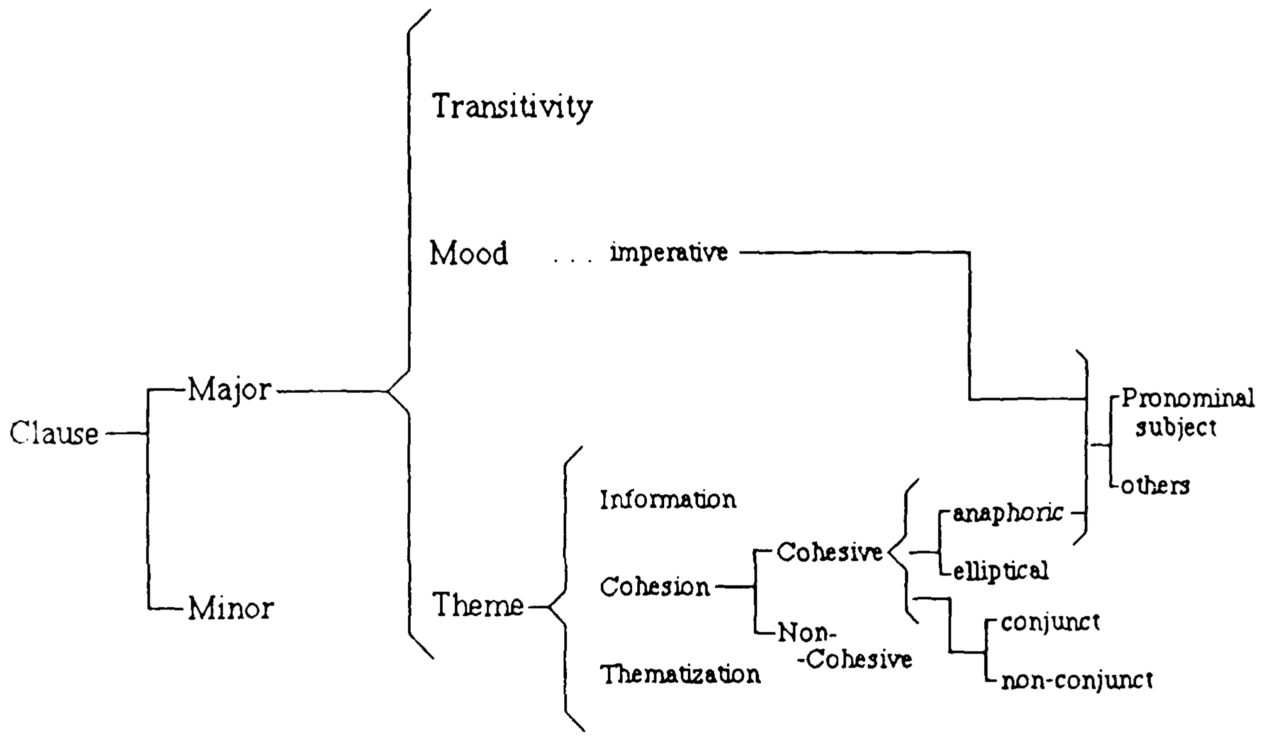
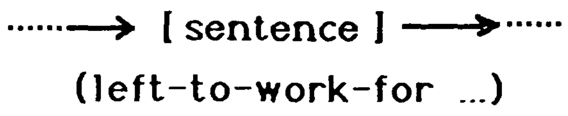


Figure Two

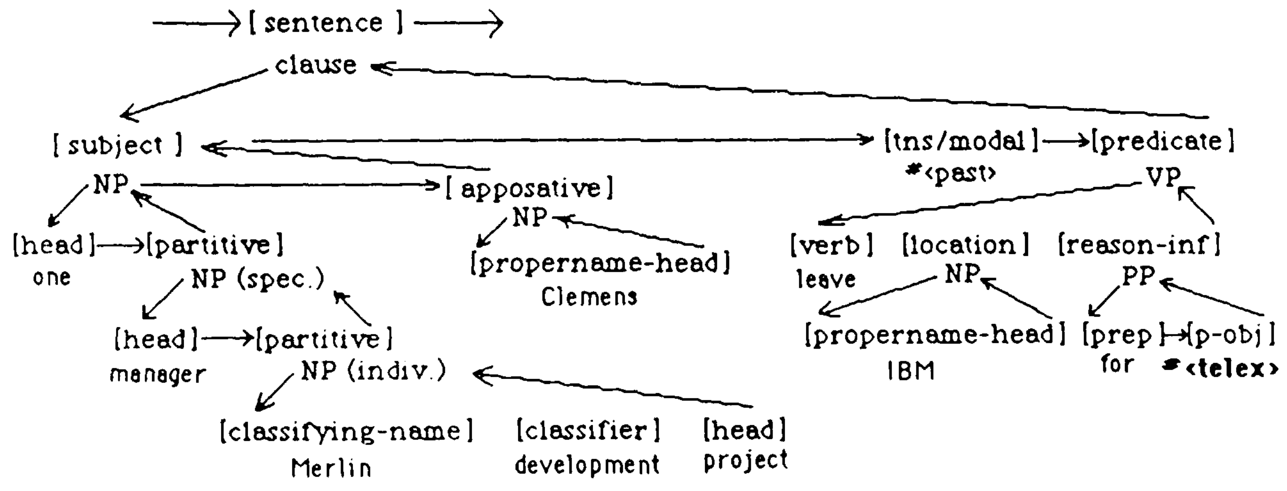
```
┌                                                              ┐
 CAT = NP
 PAT = (  . <DET><PREMODS><HEAD><POSTMODS> ...)
 AGR = <HEAD AGR>
  ⎧ ┌HEAD =  [ CAT = N ]                          ┐  ⎫
  ⎪ │    ⎧ ┌TYPE  = PROPER┐ ⎫                     │  ⎪
  ⎪ │    ⎨ └DET = NONE    ┘ ⎬                     │  ⎪
  ⎪ └    ⎩TYPE = COMMON     ⎭                     ┘  ⎪
  ⎨ ┌  ⎧HEAD = [ CAT = PRO ]     ⎫  ┐                ⎬
  ⎪ │  ⎩HEAD = [ CAT = SCOMP]    ⎭  │                ⎪
  ⎪ │ DET = NONE                    │                ⎪
  ⎪ │ PREMODS = NONE                │                ⎪
  ⎩ └ POSTMODS = NONE               ┘                ⎭
  ⎧PREMODS = NONE             ⎫
  ⎩PREMODS = [ CAT = ADJP ]   ⎭
  ⎧POSTMODS = NONE            ⎫
  ⎨POSTMODS = [ CAT = PP ]    ⎬
  ⎩POSTMODS = [ CAT = SREL ]  ⎭
└                                                              ┘
```

Figure Three



The first item of the message in a toplevel position of the surface structure annotated as a 'sentence'

Figure Five



Said so far:

" One of the managers of the Merlin development project, Clemens, left IBM for //"

Figure Six

END

2-81

DTIC